DOCUMENTATIE

TEMA NUMARUL 3

NUME STUDENT: COSTEA PAUL

GRUPA: 30229

**CUPRINS**

1. Obiectivul temei ........................................................................................................................... 3

2. Analiza problemei, modelare, scenarii, cazuri de utilizare ............................................................................................................................4

3. Proiectare ............................................................................................................................ 5

4.Implementare

............................................................................................................................8

5. Rezultate .......................................................................................................................... 10

6. Concluzii ............................................................................................................................10

7. Bibliografie ...........................................................................................................................12

**1. OBIECTIVUL TEMEI**

1. Obiectivul principal al temei: Dezvoltarea unei aplicații de gestionare a comenzilor pentru un depozit, utilizând baze de date relaționale și arhitectura stratificată.
2. Obiective secundare:

* Proiectarea bazei de date: Crearea a cel puțin trei tabele pentru Client, Product și Order. (Detaliat în capitolul "Proiectarea bazei de date")
* Implementarea claselor model: Crearea claselor pentru reprezentarea modelelor de date. (Detaliat în capitolul "Implementarea claselor model")
* Implementarea claselor de logică a afacerii: Crearea claselor pentru operațiile cu clienți și produse. (Detaliat în capitolul "Implementarea claselor de logică a afacerii")
* Implementarea claselor de prezentare: Crearea interfețelor grafice pentru operațiile cu clienți, produse și crearea comenzilor. (Detaliat în capitolul "Implementarea claselor de prezentare")
* Implementarea claselor de acces la date: Crearea claselor pentru accesul la baza de date și operațiile CRUD. (Detaliat în capitolul "Implementarea claselor de acces la date")
* Funcționalitatea de creare a comenzilor: Permite utilizatorului să creeze și să gestioneze comenzile, inclusiv verificarea stocului disponibil. (Detaliat în capitolul "Funcționalitatea de creare a comenzilor")
* Utilizarea tehnicilor de reflexie: Generarea dinamică a antetului tabelului și popularea acestuia cu date. (Detaliat în capitolul "Utilizarea tehnicilor de reflexie")
* Documentație de calitate: Elaborarea unei documentații complete și coerente care acoperă toate aspectele aplicației. (Detaliat în capitolul "Documentație")

**2. ANALIZA problemei, modelare, scenarii, cazuri de utilizare**

**Cerințe funcționale:**

* Aplicația trebuie să permită unui angajat să adauge un nou client.
* Aplicația trebuie să permită unui angajat să adauge un nou produs.
* Aplicația trebuie să permită unui angajat să vizualizeze toți clienții.
* Aplicația trebuie să permită unui angajat să vizualizeze toate produsele.
* Aplicația trebuie să permită unui angajat să creeze o comandă nouă.
* Aplicația trebuie să actualizeze stocul produsului după crearea unei comenzi.
* Aplicația trebuie să permită unui angajat să actualizeze informațiile despre client.
* Aplicația trebuie să permită unui angajat să actualizeze informațiile despre produs.
* Aplicația trebuie să permită unui angajat să șteargă un client.
* Aplicația trebuie să permită unui angajat să șteargă un produs.
* Aplicația trebuie să permită unui angajat să șteargă o comandă.

**Cerințe non-funcționale**:

* Aplicația trebuie să aibă o interfață prietenoasă și intuitivă pentru utilizatori.
* Aplicația trebuie să ofere performanță eficientă în manipularea unor cantități mari de date.
* Aplicația trebuie să asigure securitatea și integritatea datelor.
* Aplicația trebuie să fie compatibilă cu diferite sisteme de operare.
* Aplicația trebuie să aibă un design responsiv pentru a se adapta la diferite dimensiuni de ecran.
* Aplicația trebuie să ofere gestionarea adecvată a erorilor și să afișeze mesaje de eroare semnificative.
* Aplicația trebuie să fie bine documentată pentru o întreținere ușoară și îmbunătățiri viitoare.
* Aplicația trebuie să respecte standardele de codificare și cele mai bune practici.
* Aplicația trebuie să fie scalabilă pentru a putea gestiona viitoare creșteri și modificări ale cerințelor.
* Aplicația trebuie să ofere validarea corespunzătoare a datelor pentru a preveni introducerea incorectă sau inconsistentă a datelor.

**Descrierea use-case-urilor:**

**Use Case:** Adăugare produs

**Actor principal:** Angajat

**Scenariu principal de succes:**

* Angajatul selectează opțiunea de adăugare a unui client nou.
* Aplicația afișează un formular în care trebuie introduse detaliile clientului.
* Angajatul completează numele clientului, adresa și numărul de contact.
* Angajatul apasă butonul "Perform action".
* Aplicația salvează datele clientului în baza de date și afișează un mesaj de confirmare.
* Secvență alternativă: Valori invalide pentru datele clientului

**Use Case:** Vizualizare clienți

**Actor principal:** Angajat

**Scenariu principal de succes:**

* Angajatul selectează opțiunea de adăugare a unui produs nou.
* Aplicația afișează un formular în care trebuie introduse detaliile produsului.
* Angajatul completează numele produsului, prețul și stocul curent.
* Angajatul apasă butonul "Adăugare".
* Aplicația salvează datele produsului în baza de date și afișează un mesaj de confirmare.
* Secvență alternativă: Valori invalide pentru datele produsulu

**Use Case:** Creare comandă

**Actor principal:** Angajat

**Scenariu principal de succes:**

* Angajatul selectează opțiunea de creare a unei comenzi noi.
* Aplicația afișează o interfață în care angajatul poate selecta un produs existent și un client existent.
* Angajatul introduce cantitatea dorită pentru produs.
* Angajatul apasă butonul "Perform action".
* Aplicația verifică disponibilitatea stocului produsului.
  + - * + Diagrama use-case
* A diagram of a person with Ice hockey rink in the background

  Description automatically generated with medium confidence

**3. PROIECTARE**

Implementarea tehnicilor de reflexie se realizează în clasa abstractă **AbstractDAO<T>**. Această clasă furnizează metode generice pentru accesarea bazei de date, cum ar fi crearea obiectelor, inserarea, actualizarea și ștergerea acestora.

În constructorul clasei **AbstractDAO**, se utilizează reflexia pentru a obține tipul concret al entității reprezentate de clasa generică **T**. Acest lucru este realizat prin extragerea tipului din argumentul generic al clasei și este necesar pentru a putea efectua operații specifice asupra tabelei corespunzătoare entității.

Clasa **AbstractDAO** oferă metode protejate pentru gestionarea conexiunii la baza de date, închiderea conexiunii, pregătirea și executarea interogărilor SQL, precum și manipularea rezultatelor obținute. Metodele pentru execuția interogărilor SELECT, INSERT, UPDATE și DELETE sunt definite utilizând parametri variabili pentru a permite transmiterea valorilor necesare.

Pentru crearea obiectelor din rezultatele interogărilor SELECT, se utilizează reflexia pentru a instanțiați obiecte de tip **T**. Se obține constructorul corespunzător și se setează accesibilitatea acestuia pentru a permite instantierea. Apoi, pentru fiecare câmp al obiectului, se obține numele câmpului și valoarea corespunzătoare din rezultatul interogării. Prin intermediul clasei **PropertyDescriptor**, se obține metoda de scriere corespunzătoare câmpului și se apelează această metodă pentru a seta valoarea în obiectul creat. Obiectele rezultate sunt adăugate într-o listă și returnate.

Metodele **findAll()** și **findById(int id)** utilizează metoda **executeSelectQuery()** pentru a obține rezultatele interogării SELECT și apoi folosesc metoda **createObjects()** pentru a crea obiectele corespunzătoare.

Metoda **insert(T entity)** efectuează o interogare de tip INSERT pentru a adăuga un obiect în baza de date și returnează obiectul inserat.

Metoda **update(T entity)** este definită în mod abstract și trebuie implementată în clasele concrete care extind clasa **AbstractDAO**. Aceasta permite actualizarea unui obiect în baza de date.

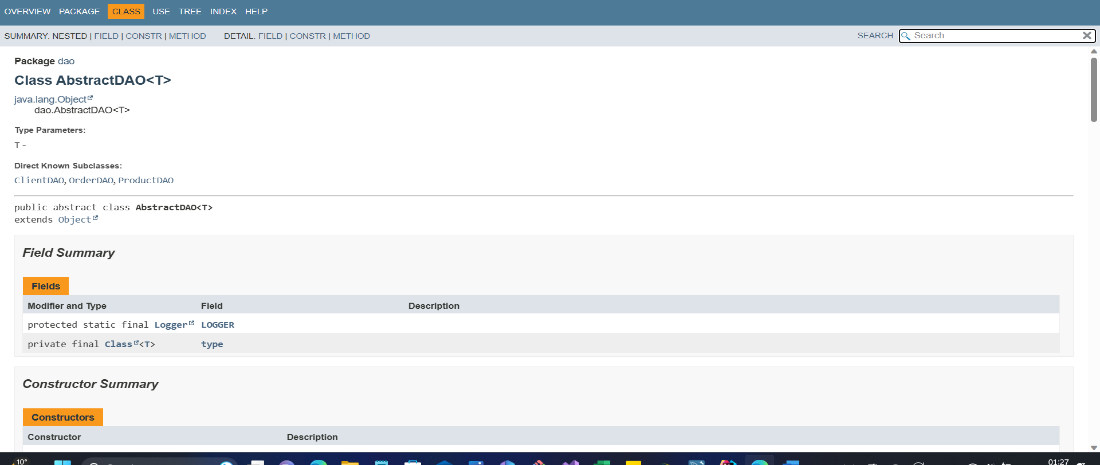
Metoda **delete(T entity)** efectuează o interogare de tip DELETE pentru a șterge un obiect din baza de date.

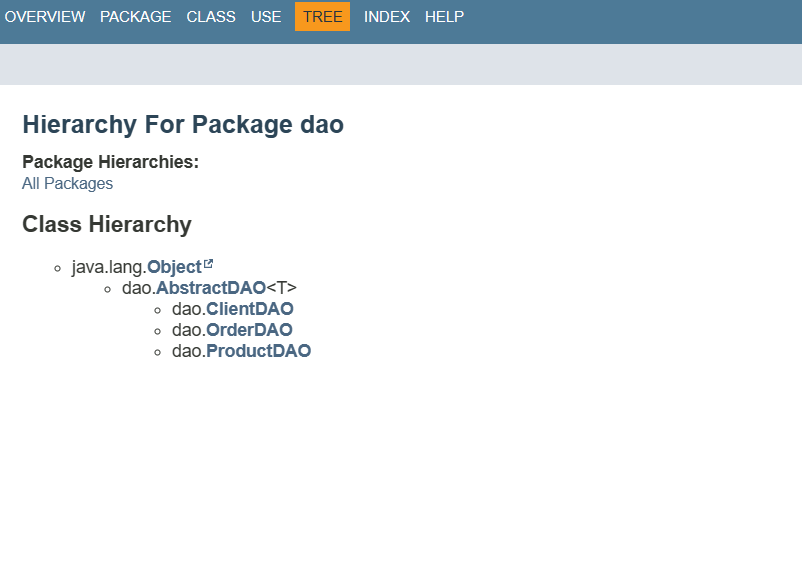
Prin intermediul utilizării tehnicilor de reflexie, clasa **AbstractDAO** devine o clasă generică reutilizabilă, care poate fi extinsă pentru a gestiona diferite entități în baza de date. Această abordare reduce duplicarea codului și facilitează adăugarea de funcționalități specifice în clasele concrete de acces la date.

Diagrama UML

A picture containing text, diagram, screenshot, font

Description automatically generated





**4. IMPLEMENTARE**

Clasa "Client" este un model pentru obiectele de tip client în aplicație și conține variabilele de instanță pentru clientId, name, contact și address, precum și metodele pentru accesarea și modificarea acestor variabile.

Clasa "Order" reprezintă un model pentru obiectele de tip comandă în aplicație și conține variabilele de instanță pentru orderId, client, product și quantity, precum și metodele pentru accesarea și modificarea acestor variabile.

Clasa "Product" reprezintă un model pentru obiectele de tip produs în aplicație și conține variabilele de instanță pentru productId, name, price și stock, precum și metodele pentru accesarea și modificarea acestor variabile.

Clasa "ConnectionFactory" este responsabilă de crearea și gestionarea conexiunii la baza de date. Aceasta utilizează driverul JDBC pentru conectarea la o bază de date MySQL locală. Clasa conține metode pentru crearea unei conexiuni, închiderea conexiunii, închiderea declarațiilor SQL și închiderea setului de rezultate.

Parametrii de configurare pentru conexiunea la baza de date sunt specificați ca constante în clasă (DRIVER, DBURL, USER și PASS). Metoda **createConnection()** este folosită pentru a crea o nouă conexiune la baza de date pe baza acestor parametri.

Clasa implementează un singleton, asigurându-se că există o singură instanță a clasei. Aceasta oferă o metodă statică **getConnection()** pentru a obține conexiunea la baza de date.

Metodele **close()** sunt utilizate pentru închiderea conexiunii, declarațiilor SQL și setului de rezultate pentru a elibera resursele și a preveni scurgeri de memorie.

Clasa "ClientDAO" este o clasă care se ocupă de accesarea și manipularea datelor pentru entitățile "Client" într-o bază de date. Aceasta oferă metode pentru a obține toți clienții, a obține un client după ID, a insera un client nou, a actualiza datele unui client existent și a șterge un client.

Clasa "OrderDAO" este responsabilă de gestionarea accesului și manipularea datelor entităților "Order" într-o bază de date. Aceasta oferă metode pentru a obține toate comenzile, a obține o comandă după ID, a insera o comandă nouă, a actualiza datele unei comenzi existente și a șterge o comandă.

Clasa "ProductDAO" este responsabilă de gestionarea accesului și manipularea datelor entităților "Product" într-o bază de date. Aceasta utilizează o conexiune la bază de date prin intermediul clasei "ConnectionFactory". Clasa oferă metode pentru a obține toate produsele, a obține un produs după ID, a insera un produs nou, a actualiza datele unui produs existent, a șterge un produs și a actualiza stocul unui produs.

Clasa "ClientView" reprezintă o interfață grafică pentru gestionarea clienților într-o aplicație. Interfața conține câmpuri de introducere pentru ID, nume, adresă și contact ale unui client, precum și butoane pentru adăugarea, actualizarea și ștergerea unui client. De asemenea, afișează o tabelă cu detalii despre clienți. Clasa utilizează obiecte din clasele "ClientDAO" pentru a accesa și manipula datele clienților în baza de date, iar clasa "ConnectionFactory" pentru a obține conexiunea la bază de date.

Clasa "OrderGUI" reprezintă o interfață grafică pentru gestionarea comenzilor într-o aplicație. Interfața conține câmpuri de introducere pentru ID-ul comenzii, ID-ul clientului, ID-ul produsului și cantitatea comenzii, precum și butoane pentru adăugarea, actualizarea și ștergerea unei comenzi. De asemenea, afișează o tabelă cu detalii despre comenzile existente. Clasa utilizează obiecte din clasele "OrderDAO" și "ProductDAO" pentru a accesa și manipula datele comenzilor și produselor în baza de date. Există o referință către clasa "ProductView" pentru a actualiza afișarea tabelului de produse atunci când se fac modificări în comenzile care afectează stocul produselor.

Clasa ProductGUI reprezintă o componentă a interfeței grafice a utilizatorului (GUI) în Java Swing pentru gestionarea produselor. Aceasta include câmpuri de text pentru informațiile produsului, butoane pentru adăugarea, actualizarea și ștergerea produselor, precum și o tabelă pentru afișarea datelor despre produse. Clasa interacționează cu un obiect ProductDAO pentru a efectua operațiuni în baza de date. Constructorul stabilește GUI-ul, inițializează componentele necesare și populează tabela de produse. Metodele addProduct(), updateProduct() și deleteProduct() se ocupă de acțiunile utilizatorului pentru adăugarea, actualizarea și ștergerea produselor.

Clasa Main servește ca punct de intrare în aplicație. Aceasta creează și afișează interfața grafică a utilizatorului (GUI) prin apelul metodei createAndShowGUI().

În metoda createAndShowGUI(), sunt create instanțe ale claselor ClientView, OrderView și ProductGUI. Acestea sunt principalele componente ale GUI-ului aplicației.

Apoi, se creează un JFrame cu titlul "Order Management". Aspectul frame-ului este setat ca un GridLayout cu o singură linie și trei coloane. Cele trei componente GUI (clientGUI, orderGUI și productGUI) sunt adăugate frame-ului folosind metoda getContentPane() corespunzătoare fiecăreia.

În cele din urmă, frame-ul este ajustat pentru a se potrivi dimensiunii preferate a componentelor sale și este setat ca vizibil, afișând GUI-ul utilizatorului.

**5. Rezultate**  
 Am rulat si verificat aplicatia de nunumarate ori pentru fiecare use-case si scenariu,testandu-i toate feature-urile pe care le are atat in baza de date cat si in interfata

**6.CONCLUZII**

**Ce am învățat din temă?**

* MySQL&Java impreuna,in special baze de date
* Noul degisn conceptual al programului
* Sa utilizez tehnici de reflexie
* Sa generez Javadoc

**Dezvoltări posibile:**

* Sa primeasca prin metoda de contact clientul cu confirmarea comezii fie aceasta trimisa,anulata sau modificata
* Eficientizare metodelor
* Aspectul

**Concluzie**

Proiectul implementează o aplicație de management a produselor cu un design bazat pe programarea orientată pe obiecte. S-au respectat practici precum limitele de dimensiune ale claselor și metodelor, utilizarea convențiilor de denumire în Java și documentarea codului folosind Javadoc, cu generarea fișierelor JavaDoc corespunzătoare.

S-a utilizat o bază de date relațională pentru stocarea datelor aplicației, având cel puțin trei tabele: Client, Product și Order. Interfața grafică include ferestre dedicate operațiilor cu clienți și produse, care permit adăugarea, modificarea și ștergerea acestora, precum și afișarea lor în tabele (JTable). De asemenea, a fost creată o fereastră pentru crearea comenzilor de produse, în care utilizatorul poate selecta un produs existent, un client existent și poate introduce cantitatea dorită pentru a crea o comandă validă. În cazul în care nu există suficiente produse în stoc, se afișează un mesaj de sub-stocare. După finalizarea comenzii, stocul produsului este actualizat prin decrementare.

S-a utilizat tehnica de reflexie pentru a crea o metodă care primește o listă de obiecte și generează antetul tabelului, extrăgând prin reflexie proprietățile obiectului și populând tabelul cu valorile elementelor din listă.

Documentația proiectului este de bună calitate și acoperă secțiunile din șablonul de documentație. Arhitectura stratificată a fost implementată, cu cel puțin patru pachete: dao, connection, model și presentation.

În cadrul pachetului model, a fost definită o clasă Bill imutabilă utilizând înregistrările Java (Java records). Un obiect Bill este generat pentru fiecare comandă și este stocat într-o tabelă de logare (Log). Facturile pot fi doar inserate și citite din tabela Log, nefiind permise actualizări.

De asemenea, s-au utilizat tehnici de reflexie pentru a crea o clasă generică care conține metode pentru accesarea bazei de date (cu excepția tabelei Log): creare obiect, editare obiect, ștergere obiect și căutare obiect. Interogările pentru accesarea bazei de date pentru un obiect specific care corespunde unei tabele sunt generate dinamic prin reflexie.

Proiectul îndeplinește cerințele și utilizează o varietate de tehnici și concepte, precum programarea orientată pe obiecte, baze de date relaționale, interfață grafică, reflexie și documentare de calitate

**7.BIBLIOGRAFIE**

* [FUNDAMENTAL PROGRAMMING TECHNIQUES (dsrl.eu)](https://dsrl.eu/courses/pt/materials/PT2023_A3_S1.pdf)
* [FUNDAMENTAL PROGRAMMING TECHNIQUES (dsrl.eu)](https://dsrl.eu/courses/pt/materials/PT2023_A3_S2.pdf)
* Laboratoarele+materialele domnului profesor Ion Giosan ,invatate la Programrea Orientata pe Obiecte
* StackOverflow
* Assigmentul 1
* Assigmentul 2
* 